NONLINEAR OPTIMIZATION

Qingsha Cheng 程庆沙

Matlab Optimization Toolbox

Introduction: what is optimization toolbox?

Optimization functions covered in this lecture

Optimization options

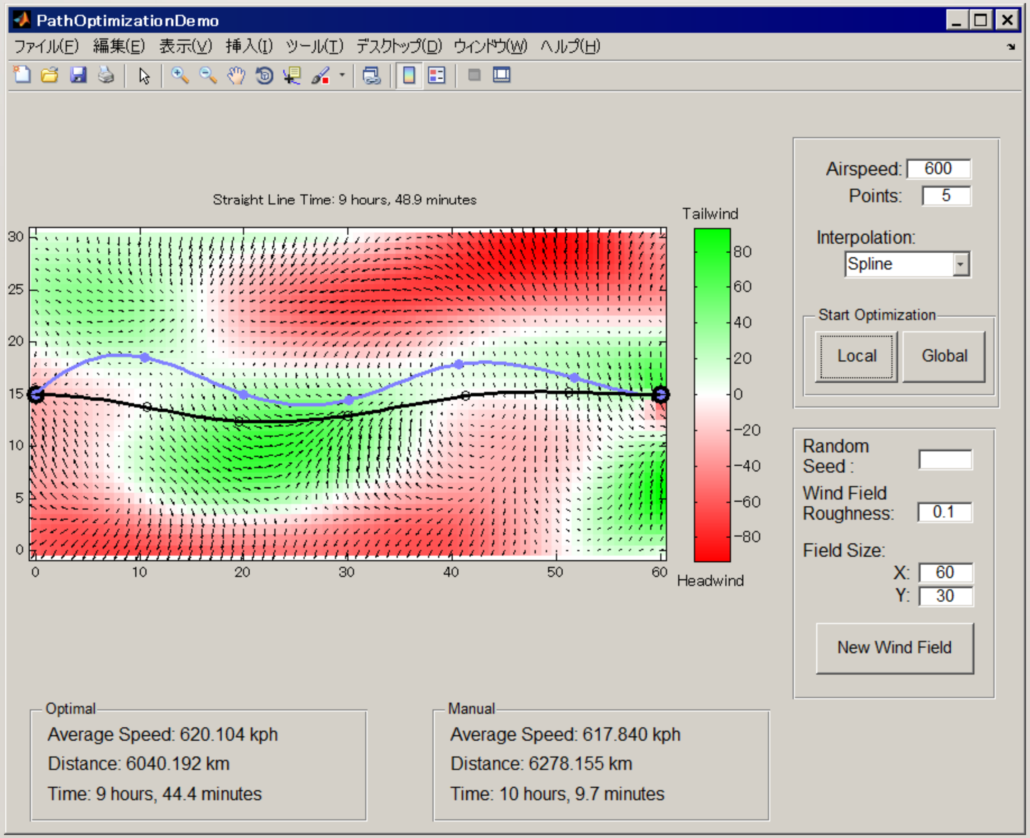
Objective function and constraints

Calling optimization routines

Examples

Matlab Optimization Toolbox

Video



What is Optimization Toolbox?

Optimization Toolbox is a set of routines that allow solving many types of optimization problems, including:

1. Unconstrained and constrained nonlinear minimization
2. Quadratic and linear programming
3. Nonlinear curve fitting
4. Solving nonlinear systems of equations
5. Solving large-scale problems

Here, we will focus on functions for **unconstrained and constrained minimization** as well as **nonlinear curve fitting**

Selected Minimization Routines

fminbnd: finds a minimum of a function of one variable on a fixed interval

fminunc: finds a minimum of an unconstrained multivariable function

fminsearch: finds a minimum of an unconstrained multivariable function (uses simplex algorithm)

fmincon: finds a minimum of a constrained multivariable function

fminimax: solves a minimax optimization problem

linprog: solves a linear programming problem

quadprog: solves a quadratic programming problem

lsqnonlin: solves nonlinear least-squares problem

Solving Optimization Problems with Optimization Toolbox

A general procedure of solving optimization problem with Optimization Toolbox is the following:

1. Specify and implement the objective function

2. Specify and implement constraints

3. Select the optimization routine

4. Set up optimization options

5. Call the optimization routine and find the solution

Optimization Options

There is a number of options available to control the optimization routines of the Optimization Toolbox; not all of them are used by all routines

Optimization options are provided to the optimization routine using an input argument *options*, which is a structure of the form:

```
options =
        Option1: Value1
        Option2: Value2
        Option3: Value3
        …
```

Optimization Options

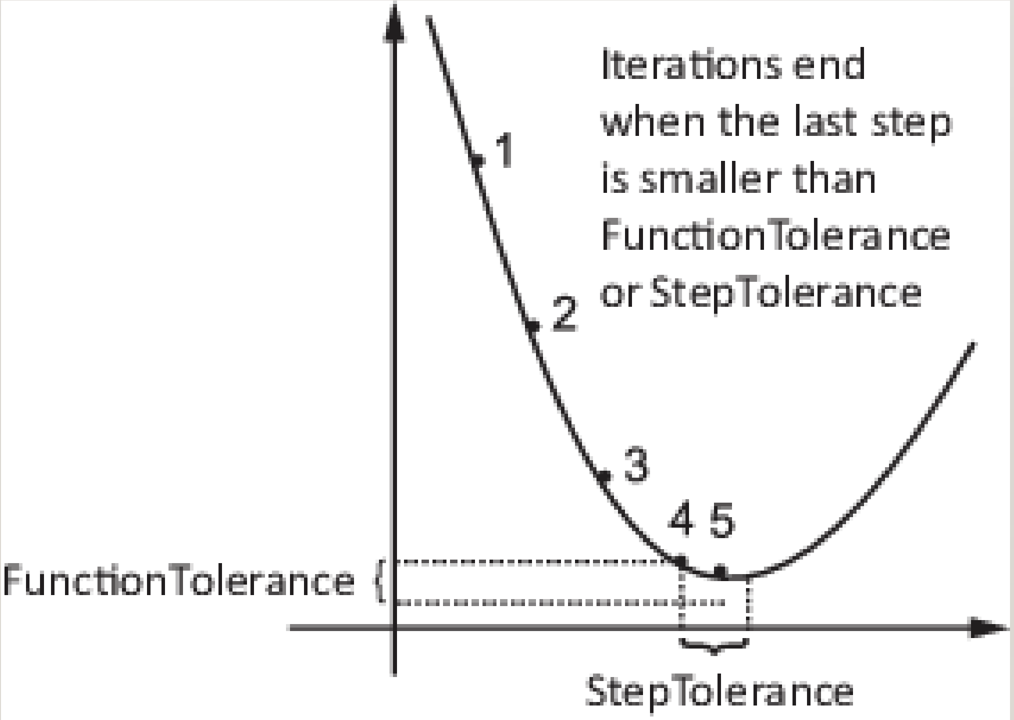
Common options:

Display	'off' | 'iter' | 'final' | 'notify'	% level of display
GradObj	'on' | {'off'}	% user-defined gradient of the objective function
GradConstr	'on' | {'off'}	% user-defined gradient of nonlinear constraints
Jacobian	'on' | {'off'}	% user-defined Jacobian of the objective function
MaxFunEvals	Positive integer	% maximum number of function evaluations allowed
MaxIter	Positive integer	% maximum number of iterations allowed
OutputFcn	User defined function	% specify a user-defined function that is called at each iteration
TolCon	Positive scalar	% termination tolerance on the constraint violation
TolFun	Positive scalar	% termination tolerance on the function value
TolX	Positive scalar	% termination tolerance on the function arguments
Hessian	'on' | {'off'}	% user-defined Hessian of objective function
DiffMaxChange	Positive scalar | {1e-1}	% maximum change in variables for finite differences
DiffMinChange	Positive scalar | {1e-8}	% minimum change in variables for finite differences

Optimization Options

Common options:

TolFun	Positive scalar	% termination tolerance on the function value
TolX	Positive scalar	% termination tolerance on the function arguments



Optimization Options

The following functions can be used to handle the optimization options structure:

optimset: create or alter the options structure

```
options = optimset('Param1',Value1, 'Param2',Value2,…)      % creates an optimization options
                                                                structure with the named
                                                                parameters having specified
                                                                values; unnamed parameters
                                                                take default values ([ ])
```

optimget: extracts the value of the named parameter

```
value = optimget(options,'Param')              % extracts the value of parameter 'Param'

value = optimget(options,'Param',DEFAULT)      % extracts the value of parameter 'Param'
                                                 and return DEFAULT is the named
                                                 parameter is not specified (is [ ])
```

Objective Function

The function to be minimized must accept an input vector *x* and **return a scalar/vector** *f* (the value of the objective function):

```
function f = fun(x)
f = …     % code for computing the function value at x
```

Example: implement function $f(x) = x_1^2+4(x_2^2-x_1)^2$

```
function f = fun(x)
f = x(1)^2+4(x(2)^2-x(1))^2;
```

Objective Function

If the gradient of *f* can be also computed and *GradObj* option is 'on', the function must also return the **gradient** in the output argument:

```
function [f,g] = fun(x)
f = …     % code for computing the function value at x
if nargout >1
   g = … % if fun is called with two output arguments, compute gradient at x
end
```

Example: implement function $f(x) = x_1^2+4(x_2^2-x_1)^2$

```
function [f,g] = fun(x)
f = x(1)^2+4*(x(2)^2-x(1))^2;
if nargout >1
   g = [2*x(1)-8*(x(2)^2-x(1))  16*(x(2)^2-x(1))*x(2)]';
end
```

Objective Function

If the Hessian matrix of f can also be computed and *Hessian* option is ‘on’, the function must also **return Hessian** in the output argument:

```
function [f,g,h] = fun(x)
f = …       % code for computing the function value at x
if nargout > 1
   g = … % if fun is called with two output arguments, compute gradient at x
   if nargout > 2
      h = … % fun is called with three output arguments, compute Hessian at x
   end
end
```

Example: implement function $f(x) = x_1^2+4(x_2^2-x_1)^2$

```
function [f,g,h] = fun(x)
f = x(1)^2+4*(x(2)^2-x(1))^2;
if nargout >1
   g = [2*x(1)-8*(x(2)^2-x(1))  16*(x(2)^2-x(1))*x(2)]’;
   if nargout > 2,  h = [10  -16*x(2); -16*x(2)  48*x(2)-16*x(1)];  end
end
```

Objective Function

Any additional arguments of f can be provided using a variable argument list; these arguments will be passed through a variable argument list of an optimization routine:

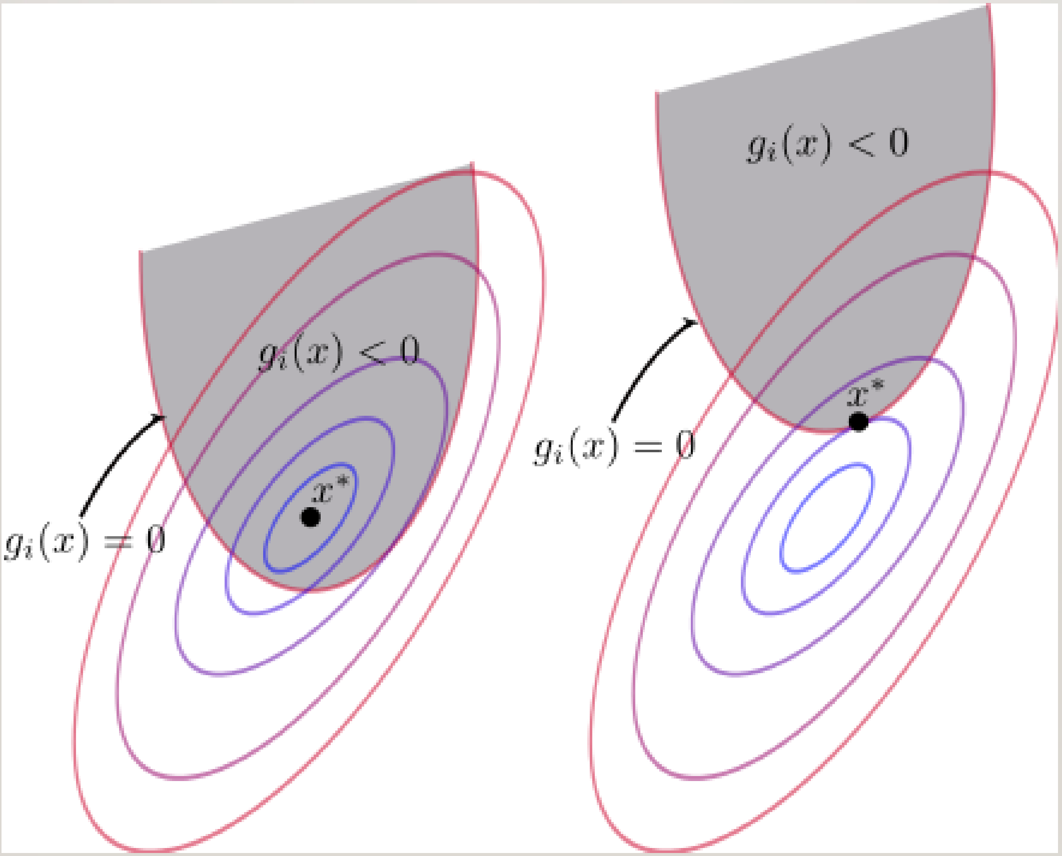
```
function f = fun(x,varargin)
```

Example: implement $f(x) = a \cdot \exp(-\lambda x)$ with a and λ as parameters; use $a = 1$ and $\lambda = 1$ in case no parameters are provided.

```
function f = fun(x,varargin)
alpha = 1;
lambda = 1;
if nargin > 1
    alpha = varargin{1};
end
if nargin > 2
    lambda = varargin{2};
end
f = alpha*exp(-lambda*x);
```

Constraints

Optimization routines included in the Optimization Toolbox can handle (depending on the routine) constraints:



Constraints

Optimization routines included in the Optimization Toolbox can handle (depending on the routine) the following types of constraints:

1. Lower and upper bounds of the form: $\boldsymbol{lb} \le \boldsymbol{x} \le \boldsymbol{ub}$
2. Linear inequality constraints of the form: $\boldsymbol{Ax} \le \boldsymbol{b}$
3. Linear equality constraints of the form $\boldsymbol{A_{eq}x} = \boldsymbol{b_{eq}}$
4. Nonlinear inequality constraints of the form $\boldsymbol{c}(\boldsymbol{x}) \le 0$
5. Nonlinear equality constraints of the form $\boldsymbol{c_{eq}}(\boldsymbol{x}) = 0$

Parameters of **bounds and linear constraints** are passed to the optimization routine using corresponding **vectors/matrices**

Nonlinear constraints are provided using a **function handle** implementing the constraint function(s)

Nonlinear Constraints

The function that computes nonlinear constraints must accept an input vector x and return two vectors c and ceq; vector c contains the nonlinear inequalities, while vector ceq contains nonlinear equalities, both evaluated at x:

```
function [c,ceq] = nonlcon(x)
c = …               % code for computing nonlinear inequalities at x
ceq = …             % code for computing nonlinear equalities at x
```

Example: implement constraints $x_1^2 + x_2^2 = 1$, $x_1 \leq 2x_2^2$, $x_2(1+x_1^2) \geq 2$

```
function [c,ceq] = nonlcon(x)
c(1) = 2*x(2)^2-x(1);
c(2) = 2-x(2)*(1+x(1)^2);
ceq = x(1)^2+x(2)^2-1;
```

Nonlinear Constraints

If the gradient of *c* and *ceq* can be also computed and *GradConstr* option is ‘on’, the function must also return the gradients of the nonlinear constraints in the output argument:

```
function [c,ceq,gc,gceq] = fun(x)
c = …                  % code for computing nonlinear inequalities at x
ceq = …                % code for computing nonlinear equalities at x
if nargout > 2
    gc = …             % code for computing gradient of nonlinear inequalities at x
    gceq = …           % code for computing gradient of nonlinear equalities at x
end
```

Calling Optimization Routine

The general syntax of calling optimization routines in Optimization Toolbox if the following (details may vary for specific routines):

[x,fval,exitflag,*other_output_arguments*] = *optimization_routine*(fun,x0,*constraint_data*,… options,*additional_arguments*)

Input arguments:

optimization_routine: name (handle) of the optimization routine
fun: handle to the objective function
x0: starting point
constraint_data: data determining constraint functions (routine dependent)
options: optimization options structure
additional_arguments: additional parameters of the objective function (comma-separated)

Output arguments:

x: final result
fval: objective function value at final result
exitflag: integer identifying the reason for algorithm termination
other_output_arguments: additional output data (e.g. gradien of the objective function at x)

Calling Optimization Routine: Examples

```
[x,fval] = fminbnd(fun,x1,x2,options)
[x,fval] = fminunc(fun,x0,options)
[x,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
[x,fval] = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
[x,fval] = linprog(fun,x0,A,b,Aeq,beq,lb,ub,options)
[x,fval] = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
[x,resnorm,residual] = lsqnonlin(fun,x0,lb,ub,options)
```

Remarks:

1. x_1 and x_2 determine the end of the search interval
2. *A*, *b*, *Aeq*, *beq*, *lb*, *ub* are vectors and matrices determining lower/upper bounds and inequality/equality linear constraints (use empty array [ ] if any of the constraints is not set up); *nonlcon* is a handle of the function calculating nonlinear constraints (use empty string '' in case there are no nonlinear constraints)
3. *resnorm* and *residual* are norm and value of the objective function at *x*

Using Output Function

Optimization routines in the Optimization Toolbox allow the user to call a special function, *output function*, at each iteration of the optimization algorithm

This feature is enabled by setting optimization option *OutputFcn* to be a handle of the output function, e.g.,

```
options = optimset('OutputFcn',@outfun)
```

Typical use of an output function would be to

- **visualize** the optimization process (e.g., plotting the points at each iteration)
- **use customized stopping criteria**

Using Output Function

Definition of the output function:

```
function stop = outfun(x,optimValues,state)
```

where *x*: point computed by the algorithm at the current iteration

optimValues: structure containing data from the current iteration (e.g., function value at *x*)

state: current state of the algorithm (e.g., 'init' – initial state before iteration, 'iter' – algorithm is at the end of an iteration)

stop: flag that is *true* or *false*, depending on whether the routine should quit or continue

Using Output Function

Example 1: stopping the optimization if the function value is smaller than -0.5 at the end of a current iteration

```
function stop = outfun(x,optimValues,state)
stop = false;
if strcmp(state,'iter') && optimValues.fval < -0.5
    stop = true;
end
```

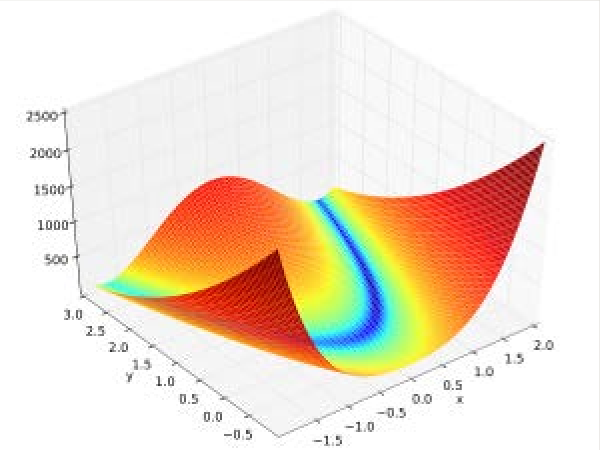
Example 2: plotting optimization path

```
function stop = outfun(x,optimValues,state)
global x_prev;
if strcmp(state,'iter')
    if ~isempty(x_prev), plot([x(1) x_prev(1)],[x(2) x_prev(2)],'k');  end
    x_prev=x;
end
```

Example 1: Minimizing Rosenbrock Function

Objective function

```
function f = rosenbrock(x)
f = (1-x(1))^2+(x(2)-x(1)^2)^2;
```



Optimization options

```
options = optimset('Display','iter','TolX',1e-8,'TolFun',1e-8);
```

Calling optimization routine (we use *fminunc* here)

```
x0 = [0 0]';
options = optimset('Display','iter','TolX',1e-8,'TolFun',1e-8);
[x,f] = fminunc(@rosenbrock,x0,options);
disp(['final result: [',num2str(x'),']',]);
disp(['final function value: ',num2str(f)]);
```

Example 1: Minimizing Rosenbrock Function

Results:

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	1		2
1	12	0.771192	0.0817341	5.34
2	15	0.610657	1	6.73
3	18	0.52245	1	7.11
4	24	0.261629	0.707501	1.88
5	30	0.248996	0.5	3.44
6	33	0.207485	1	2.94
7	36	0.125351	1	1.5
8	39	0.0893497	1	3.93
9	42	0.0308666	1	1.23
10	48	0.0200762	0.322024	1.95
11	51	0.0138484	1	1.57
12	54	0.0044155	1	0.303
13	60	0.00268685	0.5	1.14
14	63	0.000276581	1	0.28
15	66	4.21112e-005	1	0.122
16	69	1.37268e-006	1	0.00794
17	72	7.47901e-007	1	0.0303
18	75	3.36274e-009	1	0.00222
19	78	6.74271e-011	1	7.24e-005

Iteration	Func-count	f(x)	Step-size	First-order optimality
20	81	1.94706e-011	1	1.07e-006

Line search cannot find an acceptable point along the current search direction.

final result: [1 0.99999]

final function value: 1.9471e-011

Example 2: Minimizing Quadratic Function with Constraints

Task: minimize $f(x) = x_1^2 + x_2^2$ with linear constraint $\boxed{x_2 \geq -x_1 + 2}$ and nonlinear constraint $\boxed{x_2^2 \leq x_1 - 1.}$

Objective function

```
function f = quadratic(x)
f = x(1)^2 + x(2)^2;
```

Nonlinear constraint function

```
function [c,ceq] = quadratic_constr(x)
ceq = [ ];
c(1) = -x(1) + x(2)^2 + 1;
```

Example 2: Minimizing Quadratic Function with Constraints

Optimization options

```
options = optimset('Display','iter','TolX',1e-8,'TolFun',1e-8);
```

Calling optimization routine (we use *fmincon* here)

```
x0 = [10 0]';
options = optimset('Display','iter','TolX',1e-8,'TolFun',1e-8);
[x,f] = fmincon(@quadratic,x0,[-1 -1],[-2],[ ],[ ],[ ],[ ],@quadratic_constr,…
    options);
disp(['final result: [',num2str(x'),']',]);
disp(['final function value: ',num2str(f)]);
```

$$\boxed{x_2 \geq -x_1 + 2}$$

```
[x,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Example 2: Minimizing Quadratic Function with Constraints

Results:

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	First-order optimality	Procedure
0	3	100	-8				
1	7	30.5	-4	0.5	-98	42.5	
2	10	3.41095	-1.332e-015	1	-13.6	4.2	
3	14	2.56501	-4.441e-016	0.5	-1.31	0.572	
4	18	2.40674	-4.441e-016	0.5	-0.291	0.226	
5	21	2.28834	0.005079	1	-0.108	0.0108	
6	24	2.29179	5.139e-006	1	0.00346	3.51e-006	
7	27	2.2918	5.31e-012	1	3.51e-006	1.69e-008	Hessian modified
8	30	2.2918	0	1	3.63e-012	4e-008	Hessian modified

Optimization terminated: first-order optimality measure less than options.TolFun
and maximum constraint violation is less than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):

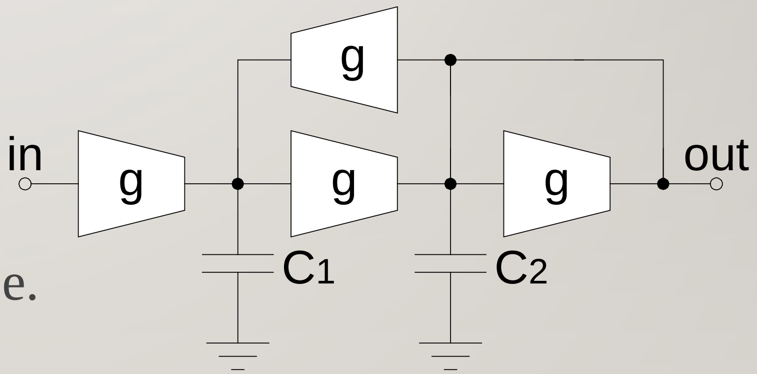
lower	upper	ineqlin	ineqnonlin
		1	1

final result: [1.382 0.61803]
final function value: 2.2918

Example 3: Optimization of Second-Order OTA-C Filter

Consider a low-pas OTA-C filter shown below. The design parameters are $x = [C_1\ C_2]^T$; g is fixed and equal 1. The design parameter domain X is given by $0.5 \le C_1, C_2 \le 1.5$. The modulus of the transfer function is given by $|H(\omega)| = g^2 \big/ \sqrt{(g^2 - \omega^2 C_1 C_2)^2 + (\omega C_1 g)^2}$ The response of the filter is a vector function $f = [f_1\ f_2\ \ldots\ f_{21}]^T$, where $f_i = |H(\omega_i)|$ with $\omega_i = 0, 0.1, \ldots, 2.0$.

Let H_0 = [1.000 1.000 0.999 0.996 0.987 0.970 0.941 0.898 0.842 0.777 0.707 0.637 0.570 0.509 0.455 0.406 0.364 0.327 0.295 0.267 0.243]^T be a target response.



Task: optimize the filter so that its response equals the target response in least-squares sense using *lsqnonlin* from Matlab Optimization Toolbox. Use *OutputFcn* option in order to plot the current filter response and the target response as well as $\|x^{(i)} - x^{(i-1)}\|$ in a lin-log scale after each iteration. Display the final result.

operational transconductance amplifiers (OTA) and capacitors (C)

Example 3: Optimization of Second-Order OTA-C Filter

Filter response function $|H(\omega)| = g^2 \Big/ \sqrt{(g^2 - \omega^2 C_1 C_2)^2 + (\omega C_1 g)^2}$

```
function H = lp_filter(x,omega)
for j=1:length(omega)
    H(j) = 1/sqrt((1-omega(j)^2*x(1)*x(2))^2+(omega(j)*x(1))^2);
end
```

Error function $\min \| |H(\omega)| - H_0 \|$

```
function E = lp_filter_lsq(x,omega,H0)
E = feval('lp_filter',x,omega) - H0;
```

Optimization options

```
options = optimset('Display','iter','OutputFcn',@lp_filter_out);
```

Example 3: Optimization of Second-Order OTA-C Filter

Output function

```
function stop = lp_filter_out(x,optimValues,state,varargin)
stop = false;
global x_prev;                %%global variable can be saved for next time use
global x_conv;                %%global variable can be saved for next time use
omega = varargin{1};
H0 = varargin{2};
f = feval('lp_filter',x,omega);
if strcmp(state,'iter')
    subplot(1,2,1);
    plot(omega,f,'b',omega,H0,'ro','LineWidth',2,'MarkerSize',6);
    grid on
    if ~isempty(x_prev)
        x_conv(length(x_conv)+1)=norm(x-x_prev);
    else
        x_conv(1) = norm(x);
    end
    x_prev = x;
    subplot(1,2,2);
    semilogy(1:length(x_conv),x_conv,'bo-','LineWidth',2,'MarkerSize',8);
    grid on
    drawnow;                  %%updates figures and processes any pending callbacks.
    pause(1);
end
```

Example 3: Optimization of Second-Order OTA-C Filter

Calling optimization routine

```
H0 = [1.000 1.000 0.999 0.996 0.987 0.970 0.941 0.898 0.842 0.777 0.707
    0.637 0.570 0.509 0.455 0.406 0.364 0.327 0.295 0.267 0.243];
omega = 0:0.1:2;
x0 = [1 1]';
options = optimset('Display','iter','OutputFcn',@lp_filter_out);
lb = [0.5 0.5]';
ub = [1.5 1.5]';
[x,f] = lsqnonlin(@lp_filter_lsq,x0,lb,ub,options,omega,H0);
disp(['final result: [',num2str(x'),']',]);
disp(['final function value: ',num2str(f)]);
```

Example 3: Optimization of Second-Order OTA-C Filter

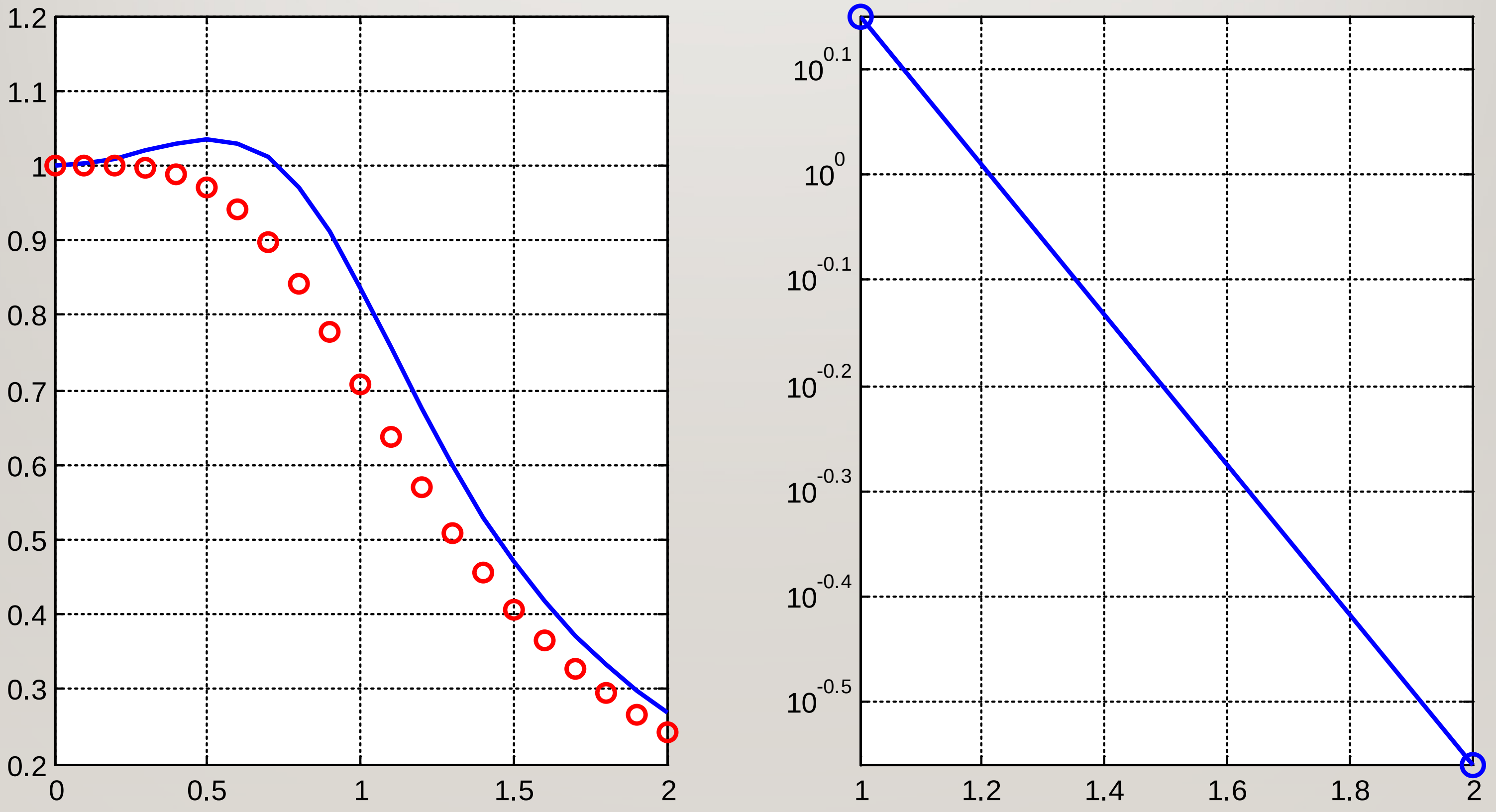
Results:

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	3	0.592207		0.913	
1	6	0.129362	0.39066	0.205	1
2	9	0.0242604	0.22668	0.0477	1
3	12	0.00325591	0.148347	0.0108	1
4	15	0.000201779	0.0830914	0.00198	1
5	18	2.87266e-006	0.028891	0.000173	1
6	21	9.50342e-007	0.00326869	1.98e-006	1
7	24	9.50084e-007	3.85063e-005	1.81e-010	1

Optimization terminated: first-order optimality less than OPTIONS.TolFun,
and no negative/zero curvature detected in trust region model.
final result: [1.4144 0.70667]
final function value: 9.5008e-007

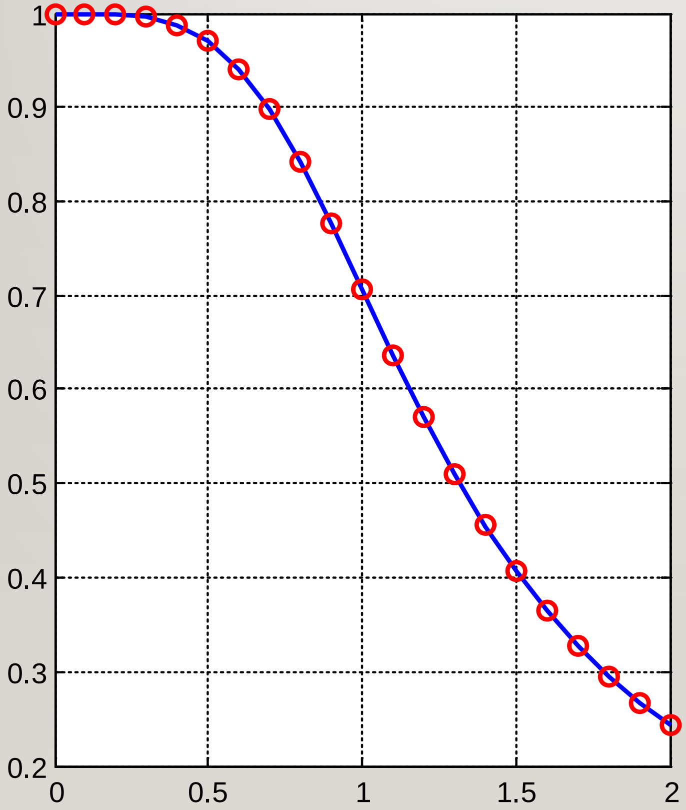
Example 3: Optimization of Second-Order OTA-C Filter

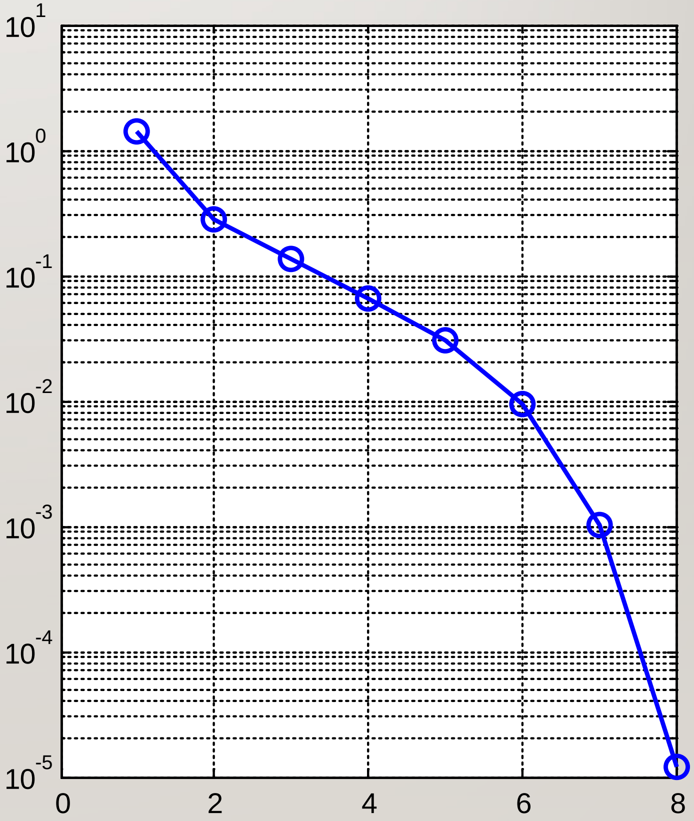
Results: plots after first iteration



Example 3: Optimization of Second-Order OTA-C Filter

Results: final plots





Bibliography

Matlab's help

Exercise 1: Unconstrained Optimization

Use *fminunc* routine to minimize a multi-dimensional generalization of the Rosenbrock function of the form

$$f(x) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \right]$$

where $x = [x_1\ x_2\ \ldots\ x_n]^T$. Consider n = 3, 5 and 10. Use zero vector as a starting point in each case.

Perform the same task using *fminsearch* routine. Compare the number of function evaluations necessary to obtain function value smaller than 10^{-10}.

Exercise 2: Constrained Optimization

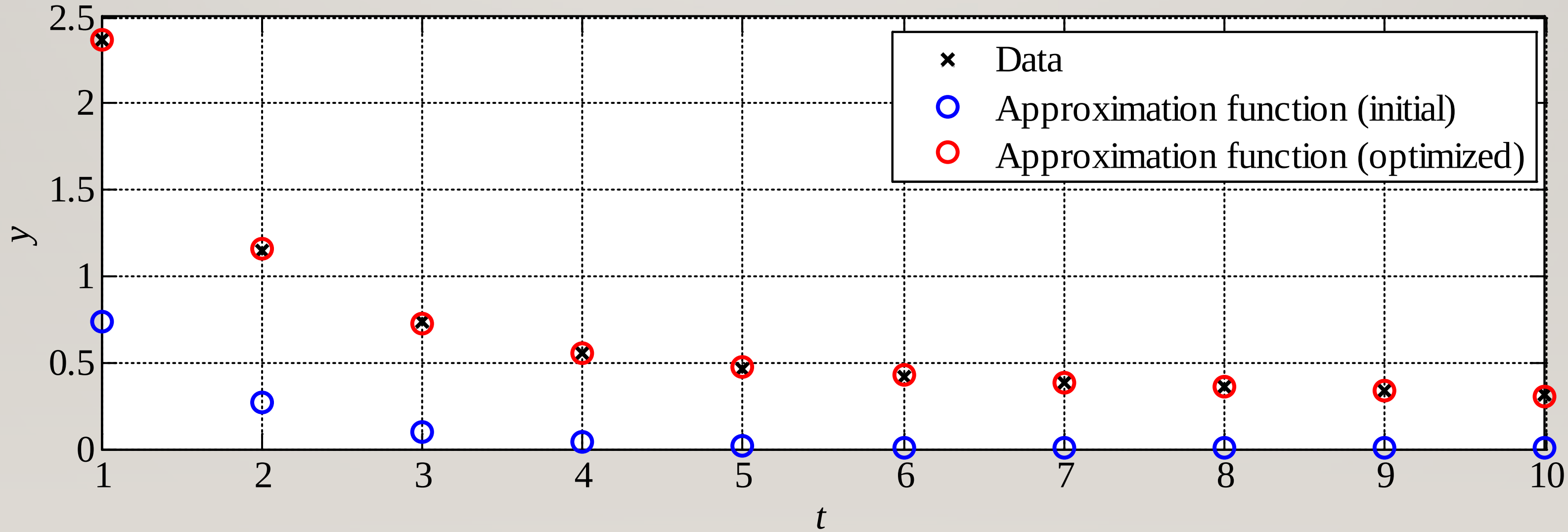
Use *fmincon* routine to minimize function $f(x) = \exp(x_1 + x_2/2 ... + x_n/n)$ so that $||x|| \leq 1$. Use zero vector as a starting point.

Perform the task for n = 2, 3, 4 and 20.

Exercise 3: Data Fitting

Use *lsqnonlin* routine from Matlab Optimization Toolbox to find the parameters a, b, c and d of the function $g(t) = a \cdot \exp(-b \cdot t) + c \cdot \exp(-d \cdot t)$ that approximates the following set of data: {(1,2.3743),(2,1.1497),(3,0.7317),(4,0.5556),(5,0.4675),(6,0.4157),(7,0.3807),(8,0.3546),(9,0.3337),(10,0.3164)} as well as possible in the least-square sense. Lower and upper bounds for parameters are 0 and 10. The initial parameter values are $a = b = c = d = 1$. Plot the data, and the approximation function for initial and optimized parameters.

Plot of the data and the approximation function (initial and optimized):



```
function R=black_box(x)
c=1;
for j=1:length(x)
    R(j)=x(j)/j+c*x(j)^2/sqrt(j);
end
```

Exercise 4: Optimization of a Black-Box Function

Consider a function *black_box* that takes a vector argument $x = [x_1 \dots x_n]^T$ and returns a vector $f(x) = [f_1(x) \dots f_n(x)]^T$. Use Matlab Optimization Toolbox to minimize the following expression $max\{f_1(x), \dots, f_n(x)\}$ under the following constraints: $x_1 + \dots + x_n = 1$, and $x_j \geq 0$ for $j = 1, \dots, n$. The starting point is $x_0 = [1/n \dots 1/n]^T$. Plot initial and optimized function values. Consider the following cases $n = 3, 5, 10$ and 20. [hint: help fminimax]

Initial (black) and optimized (red) arguments/function values for $n = 3, 5, 10$ and 20: